REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	**2. REPORT TYPE**	**3. DATES COVERED** *(From - To)*
07-02-2008	Final Report	01/15/2007 - 11/14/2007

4. TITLE AND SUBTITLE
EXTENSION OF P-ADIC EXACT SCIENTIFIC COMPUTATIONAL LIBRARY (ESCL) TO COMPUTE THE EXPONENETIAL OF RATIONAL MATRIX

5a. CONTRACT NUMBER

5b. GRANT NUMBER
FA9550-07-1-0099

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
CHAO LU

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

TOWSON UNIVERSITY
Dept. of Comp. & Info.
Sciences, 8000 York Rd
Towson, MD 21252

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
John Sjogren
875 N Randolph St. 3112
Ste 325 RM 3112
Arlington, VA 22203

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT
Distribution A: Unlimited

AFRL-SR-AR-TR-08-0132

13. SUPPLEMENTARY NOTES

14. ABSTRACT
For the past three years, we have been developing an Exact Scientific Computational Library (ESCL) using p-adic arithmetic. New algorithms have been designed and implemented for matrix operations with rational numbers by representing numerator and denominator with arbitrary length integers, all integers and fractional numbers are represented by p-adic sequences, and all arithmetic calculations are carried out in p-adic domain. In this project, we have worked on: 1) investigating the relation of the length M of p-adic expansion for a rational matrix and the periodicity of a resulted p-adic sequence from arithmetic operation in p-adic field; and extension of the ESCL to compute: 2) the complex rational matrix; 3) the exponential of a rational matrix.

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**			**19b. TELEPHONE NUMBER** *(include area code)*

Extension of *P*-adic Exact Scientific Computational Library (ESCL) to

Compute the Exponential of Rational Matrix

~~Annual~~ Report (Grant #* FA9550-07-1-0099*)

Final

Chao Lu
Computer & Information Sciences Department
Towson University

February 7, 2008

TOWSON
UNIVERSITY

20080331063

1. Summary

For the past three years, we have been developing an Exact Scientific Computational Library (ESCL) using p-adic arithmetic. New algorithms have been designed and implemented for matrix operations with rational numbers by representing numerator and denominator with arbitrary length integers, all integers and fractional numbers are represented by p-adic sequences, and all arithmetic calculations are carried out in p-adic domain. In this project, we have worked on: 1) investigating the relation of the length M of p-adic expansion for a rational matrix and the periodicity of a resulted p-adic sequence from arithmetic operation in p-adic field; and extension of the ESCL to compute: 2) the complex rational matrix; 3) the exponential of a rational matrix.

2. Progress on Length *M* and Periodicity of a *p*-adic Expansion

To determine what is the efficient length M of the p-adic expansion for a rational number in the matrix operation is not a trivial problem. Let us observe what happens after the arithmetic operations of two p-adic sequences.

All rational numbers can be uniquely written in the form,

$$a = \sum_{j=0}^{\infty} a_j p^j . \qquad (1)$$

We know that a real number is rational if and only if its decimal expansion is periodic. Similarly, a p-adic number is rational if and only if its p-adic expansion is periodic. Consequently, since we are primarily interested in the p-adic expansions of rational numbers, we will be dealing only with p-adic expansions which are periodic. The expansion eventually repeats to the right. That is, if a is a rational number, then it has a repeating pattern of $a_j p^j$ in its p-adic expansion, i.e., it is of the form

$$a = .A_0 \ldots A_s \overline{a_0 \ldots a_{n-1}} . \qquad (2)$$

Addition/subtraction

Assume that we have two p-adic sequences, (s<t):

$$a = .A_1 \ldots A_s \overline{a_1 \ldots a_n}$$

$$b = .B_1 \ldots B_t \overline{b_1 \ldots b_m}$$

Considering various carry digits' effects, we concluded that the maximum length of the p-adic expansion of $(a \pm b)$ is:

$$2 \times LCM(n,m) + \max(s,t) - 1 . \qquad (3)$$

Multiplication

$$a \times b = .A_1 \ldots A_s \overline{a_1 \ldots a_n} \times .B_1 \ldots B_t \overline{b_1 \ldots b_m}$$

$$= .A_1 \ldots A_s \overline{a_1 \ldots a_n} \times .B_1 \ldots B_t + .A_1 \ldots A_s \overline{a_1 \ldots a_n} \times .\overbrace{0 \ldots 0}^{t}\overline{b_1 \ldots b_m}$$

$$= \underbrace{.A_1 \ldots A_s \overline{a_1 \ldots a_n} \times .B_1 \ldots B_t}_{1} + \underbrace{.A_1 \ldots A_s \times .\overbrace{0 \ldots 0}^{t}\overline{b_1 \ldots b_m}}_{2} + \underbrace{.\overbrace{0 \ldots 0}^{s}\overline{a_1 \ldots a_n} \times .\overbrace{0 \ldots 0}^{t}\overline{b_1 \ldots b_m}}_{3}$$

Considering all the three parts separately and the various carry digits' effects, we concluded that, in multiplication, the length of periodic part of the product is:

$$LCM(m,n) \times (p^{GCD(m,n)} - 1), \qquad (4)$$

where m and n are the length of periodic part of the two multipliers, p is the prime.

The length of periodicity of the resulting p-adic sequence can be very large from (4). But if we should represent all the p-adic sequences with a complete period during all the calculations, we can definitely carry out all the arithmetic operations exactly. Further investigation is needed to find proper length M which is smaller than the periodicity of a p-adic expansion. The determination of the efficient length M of the p-adic expansion for a rational number in the matrix operation becomes more important, when the algorithms in eigensystem computation use iterative arithmetic operations.

3. Progress on the Computation of Complex Rational Matrices

We have implemented the algorithms developed for single and matrix rational number operations (addition, subtraction, multiplication and division) in the p-adic field, to compute the complex rational arithmetic operations in the p-adic field as follows:

Complex addition

$$(a+bi)+(c+di) = (a+c)+i(b+d), \qquad (5)$$

Complex subtraction

$$(a+bi)-(c+di) = (a-c)+i(b-d), \qquad (6)$$

Complex multiplication

$$(a+bi)(c+di) = (ac-bd)+i(ad+bc), \qquad (7)$$

and **Complex division**

$$\frac{a+bi}{c+di} = \frac{ac+bd}{c^2+d^2} + i\frac{bc-ad}{c^2+d^2}. \qquad (8)$$

These complex operations are needed in the process of eigenvalue computation.

4. Progress on the Computation of Matrix Exponential

4.1 Compute matrix exponential with Jordan canonical form

Matrix decomposition methods, which are to be most efficient for problems involving large matrices evaluation of e^{tA}, are those which are based on factorizations or decompositions of the matrix A. The Jordan canonical form (*JCF*) decomposition is one of the many stated in [1]. Due to its truncation error by floating-point arithmetic operations, it has not been widely used, since a single rounding error may cause multiple eigenvalues to become distinct, which can alter the entire structure of the decomposition. If we can improve the accumulated floating-point truncation error created during the iteration process of the computation by *p*-adic arithmetic wherever possible, we hope that the stability and accuracy can be improved dramatically.

Definition: The Jordan canonical form decomposition states that there exists an invertible P such that

$$P^{-1}AP = J, \qquad (9)$$

where J is a direct sum, $J = J_1 \oplus \cdots \oplus J_k$, of Jordan blocks,

$$J_i = \begin{bmatrix} \lambda_i & 1 & 0 & \cdots & 0 \\ 0 & \lambda_i & 1 & \cdots & 0 \\ \vdots & \vdots & & & \vdots \\ & & & & 1 \\ 0 & 0 & 0 & \cdots & \lambda_i \end{bmatrix} \quad (m_i - by - m_i). \qquad (10)$$

The λ_i are eigenvalues of A, here we assume that A is rational. Each Jordan block corresponds to a linearly independent eigenvector. If any of the m_i is greater than *1*, A is said to be defective. This means that A does not have a full set of n linearly independent eigenvectors. A is derogatory if there is more than one Jordan block associated with a given eigenvalue.

For a given matrix A, its Jordan canonical form J is completely determined by the maximal number of linearly independent eigenvectors of A: the number of the Jordan blocks in J is equal to the maximal number s of linearly independent eigenvectors of A, each of which is associated with a Jordan block whose order is the same as the 'rank' of the eigenvector.

Thus the number of Jordan blocks with the same eigenvalue λ is equal to the dimension of the eigen-space $E_\lambda = N(\lambda I - A)$, the number of linearly independent eigenvectors of A belonging to λ. Moreover, the sum of the orders of all Jordan blocks associated with an eigenvalue λ is equal to the multiplicity m_λ of λ.

In principle, the problem posed by defective eigensystems can be solved by resorting to the Jordan

canonical form (JCF).

If

$$A = P[J_1 \oplus \cdots \oplus J_k]P^{-1} \qquad (11)$$

is the JCF of A, then

$$e^{tA} = P\left[e^{tJ_1} \oplus \cdots \oplus e^{tJ_k}\right]P^{-1}. \qquad (12)$$

Thus, it is enough to compute e^{tJ} for a simple Jordan block J. The exponentials of the Jordan blocks Ji can be given in closed form. For example, if

$$J_i = \begin{bmatrix} \lambda_i & 1 & & 0 \\ & \ddots & \ddots & \\ & & \ddots & 1 \\ 0 & & & \lambda_i \end{bmatrix}, \qquad (13)$$

then,

$$e^{tJ_i} = e^{\lambda_i t}\begin{bmatrix} 1 & t & t^2/2! & \cdots & t^{n-1}/(n-1)! \\ 0 & 1 & t & \ddots & t^{n-2}/(n-2)! \\ & & 1 & \ddots & \\ & & & \ddots & t \\ 0 & & & & 1 \end{bmatrix}. \qquad (14)$$

It is important to know how sensitive a quantity is before its computation is attempted.

One of the most important issues in the computation of matrix exponential with *JCF* is the evolution of performing eigenvalue analysis.

4.2 Power method

Numerical analysis, at its simplest, is an iterative technique. It is a fruitful exercise to study the Power method to get an in-depth understanding of the numerical solution for eigen-values and eigenvectors.

4.2.1 Algorithm

The Power method is an iterative technique used to determine the dominant eigenvalue of a matrix—that is, the eigenvalue with the largest magnitude. By modifying the method slightly, it can also be used to determine other eigen-values. One useful feature of the Power method is that it produces not only an eigenvalue, but also an associated eigenvector. In fact, the Power method is often applies to find an eigenvector for an eigenvalue that is determined by some other means.

To apply the Power method, we assume that the $n \times n$ matrix A has n eigen-values $\lambda_1, \lambda_2, \cdots, \lambda_n$

with an associated collection of linearly independent eigenvectors $\{v^{(1)}, v^{(2)}, \cdots, v^{(n)}\}$. Moreover, we assume that A has precisely one eigenvalue, λ_1, that is largest in magnitude, so that $|\lambda_1| > |\lambda_2| \geq |\lambda_3| \geq \cdots \geq |\lambda_n| \geq 0$.

If x is any vector in $\Re^n$, the fact that $\{v^{(1)}, v^{(2)}, \cdots, v^{(n)}\}$ is linearly independent implies that constants $\beta_1, \beta_2, \cdots, \beta_n$ exist with $x = \sum_{j=1}^{n} \beta_j v^{(j)}$.

By making a series of calculations, we get

$$\mu^{(m)} = \lambda_1 \left[\frac{\beta_1 v_{p_{m-1}}^{(1)} + \sum_{j=2}^{n} (\lambda_j / \lambda_1)^m \beta_j v_{p_{m-1}}^{(j)}}{\beta_1 v_{p_{m-1}}^{(1)} + \sum_{j=2}^{n} (\lambda_j / \lambda_1)^{m-1} \beta_j v_{p_{m-1}}^{(j)}} \right], \qquad (15)$$

and

$$x^{(m)} = \frac{A^m x^{(0)}}{\prod_{k=1}^{m} y_{p_k}^{(k)}}. \qquad (16)$$

By examining Eq. (15), we see that $\lim_{m\to\infty} \mu^{(m)} = \lambda_1$. Moreover, the sequence of vectors $\{x^{(m)}\}_{m=0}^{\infty}$ converges to an eigenvector associated with λ_1.

The Power method implementation can be stated as follows:

INPUT: dimension *n*; matrix *A*; vector *x*; tolerance *TOL*; maximum number of iterations *N*.

OUTPUT: approximate eigenvalue μ; approximate eigenvector x (with $\|x\|_\infty = 1$) or a message that the maximum number of iterations was exceeded.

Step 1 Set k=1.

Step 2 Find the smallest integer p with $1 \leq p \leq n$ and $|x_p| = \|x\|_\infty$.

Step 3 Set $x = x/x_p$.

Step 4 While ($k \leq N$) do Step 5-11.

Step 5 Set $y = Ax$.

Step 6 Set $\mu = y_p$.

Step 7 Find the smallest integer p with $1 \leq p \leq n$ and $|y_p| = \|y\|_\infty$.

Step 8 If $y_p = 0$ then OUTPUT ('Eigenvector', x);

OUTPUT ('A has the eigenvalue 0, select a new vector x and restart');

STOP.

Step 9 Set $ERR = \left\| x - (y/y_p) \right\|_\infty$;

$x = y/y_p$.

Step 10 If $ERR < TOL$ then OUTPUT (μ, x);

(The procedure was successful.)

STOP.

Step 11 Set k=k+1.

Step 12 OUTPUT ('The maximum number of iterations exceeded');

(The procedure was unsuccessful.)

STOP.

As we can see, such an algorithm is designed to get the approximate eigen-values, round-off and truncation errors may be introduced by using floating point arithmetic during the computation, but the exact linear computation can avoid that kind of errors.

4.2.2 Implementation

For example, the matrix

$$A = \begin{bmatrix} -4 & 14 & 0 \\ -5 & 13 & 0 \\ -1 & 0 & 2 \end{bmatrix}$$

has eigen-values $\lambda_1 = 6$, $\lambda_2 = 3$, and $\lambda_3 = 2$, so the Power method described in this algorithm will converge. Let $x^{(0)} = (1,1,1)'$, then

$y^{(1)} = Ax^{(0)} = (10,8,1)'$,

so

$$\left\| y^{(1)} \right\|_\infty = 10, \mu^{(1)} = y_1^{(1)} = 10, \text{ and } x^{(1)} = \frac{y^{(1)}}{10} = (1,0.8,0.1)'.$$

Continuing in this manner leads to the approximations to the dominant eigenvalue 1/3.

Comparison of the results of the exact linear computation and results using floating point arithmetic.

The results using floating point arithmetic as follows:

1	0.5555555555555558	8	0.3340807174887892
2	0.40000000000000002	9	0.33370618941088748
3	0.36111111111111094	10	0.3335195530726256
4	0.34615384615384615	11	0.3334263912153359
5	0.33950617283950624	12	0.33337984928830566
6	0.3363636363636362	13	0.33335658806567126
7	0.33483483483483478	14	0.33334495988838486

15	0.33333914640810103	38	0.33333333333402604
16	0.33333623982003013	39	0.33333333333367965
17	0.33333478656401028	40	0.33333333333350645
18	0.3333340599455038	41	0.33333333333342008
19	0.33333369663862655	42	0.33333333333337667
20	0.33333351498578179	43	0.33333333333335502
21	0.33333342415950795	44	0.33333333333334414
22	0.33333337874640823	45	0.3333333333333387
23	0.33333335603986758	46	0.33333333333333592
24	0.33333334468659981	47	0.33333333333333437
25	0.33333333900996609	48	0.33333333333333359
26	0.33333333617164973	49	0.33333333333333326
27	0.33333333475249138	50	0.33333333333333326
28	0.33333333404291232	51	0.33333333333333326
29	0.33333333368812279	52	0.33333333333333326
30	0.33333333351072825	53	0.33333333333333326
31	0.33333333342203075	54	0.33333333333333326
32	0.33333333337768223	55	0.33333333333333326
33	0.33333333335550774	56	0.33333333333333326
34	0.33333333334442072	57	0.33333333333333326
35	0.33333333333887694	58	0.33333333333333326
36	0.33333333333610504	59	0.33333333333333326
37	0.33333333333471904	60	0.33333333333333326

The results of the exact linear computation using p-adic arithmetic as follows:

1:

5/9

2:

2/5

3:

13/36

4:

9/26

5:

55/162

6:

37/110

7:

223/666

8:

149/446

9:

895/2682

10:

597/1790

11:

3583/10746

12:

2389/7166

13:

14335/43002

14:

9557/28670

15:

57343/172026

16:

38229/114686

17:

229375/688122

18:

152917/458750

19:

917503/2752506

20:

611669/1835006

21:

3670015/11010042

22:

2446677/7340030

23:

14680063/44040186

24:

9786709/29360126

25:

58720255/176160762

26:

39146837/117440510

27:

234881023/704643066

28:

156587349/469762046

29:

939524095/2818572282

30:

626349397/1879048190

31:

3758096383/11274289146

32:

2505397589/7516192766

33:

15032385535/45097156602

34:

10021590357/30064771070

35:

60129542143/180388626426

36:

40086361429/120259084286

37:

240518168575/721554505722

38:

160345445717/481036337150

39:

962072674303/2886218022906

40:

641381782869/1924145348606

41:

3848290697215/11544872091642

42:

2565527131477/7696581394430

43:

15393162788863/46179488366586

44:

10262108525909/30786325577726

45:

61572651155455/184717953466362

46:

41048434103637/123145302310910

47:

246290604621823/738871813865466

48:

164193736414549/492581209243646

49:

985162418487295/2955487255461882

50:

656774945658197/1970324836974590

51:

3940649673949183/11821949021847546

52:

2627099782632789/7881299347898366

53:

15762598695796735/47287796087390202

54:

10508399130531157/31525197391593470

55:

63050394783186943/189151184349560826

56:

42033596522124629/126100789566373886

57:

252201579132747775/756604737398243322

58:

168134386088498517/504403158265495550

59:

1008806316530991103/3026418949592973306

60:

6725375443539940 69/2017612633061982206

From the above results, we can find that floating-point operation does not function very well in terms of round-off/truncation errors. The approximation of the dominant eigenvalue remain 0.33333333333333326 from loop 49, while it should be 0.33333333333333333 precisely. The results have been rounded up or down in every step, finally, the iterative process will dramatically enhance the uncertainty of the outcome. In the meantime, using *p*-adic arithmetic will get a decisive advantage since it can keep the calculation free of round-off/truncation errors.

An obvious advantage of the exact linear computation using *p*-adic arithmetic is that, we can get results as accurate as we want with the loop increasing, while floating-point arithmetic can only reach a certain precision and stay at a value no matter how many loops it runs.

Compare to the program using floating-point arithmetic in which the results remain 0.33333333333333326 from loop 49, the program using *p*-adic arithmetic can get more and more accurate after each loop, for example, in loop 59, the approximate eigenvalue is

$$1008806316530991103/3026418949592973306 \approx 0.33333333333333333366375685,$$

after running one more loop, the approximate eigenvalue is

$$6725375443539940697/20176126330619822206 \approx 0.33333333333333333349854509$$

which is closer to 1/3.

Precision is critical to the computation of the exponential of a rational matrix since the method requires a tremendous amount of operations, the truncation errors for every step will cumulate and finally make a great difference from the exact value. As a matter of fact, a little error will alter the entire structure of J and P and lead to an utterly wrong outcome, this will put the matter beyond a doubt that our "Exactly Computing" system would prove highly valuable on the computation of the exponential of a rational matrix.

4.3 Eigenvalues of a Real General Matrix

The solution of eigensystems is a fairly complicated business, almost all routines in use nowadays trace their ancestry back to routines published in Wilkinson and Reinsch's Handbook for Automatic Computation, Vol. II, Linear Algebra [2]. A public-domain implementation of the Handbook routines in FORTRAN is the EISPACK set of programs [3]. It includes the ability to solve for eigenvalues and

eigenvectors of various kinds of matrices, has been implemented. The routines we used are translations of either the Handbook or EISPACK routines.

The matrix is first balanced. Orthogonal similarity transformations are used to reduce the balanced matrix to a real upper Hessenberg matrix. The implicit double-shifted *QR* algorithm is used to compute the eigenvalues and eigenvectors of this Hessenberg matrix.

4.3.1 Balancing

The idea of balancing is to use similarity transformations to make corresponding rows and columns of the matrix have comparable norms, thus reducing the overall norm of the matrix while leaving the eigenvalues unchanged. It is recommended to always balance non symmetric matrices. It never hurts, and it can substantially improve the accuracy of the eigenvalues computed for a badly balanced matrix.

Balancing is a procedure with of order N^2 operations. The actual algorithm used is due to Osborne, as discussed in [2]. It consists of a sequence of similarity transformations by diagonal matrices. The output is a matrix that is balanced in the norm given by summing the absolute magnitudes of the matrix elements.

Note that if the off-diagonal elements of any row or column of a matrix are all zero, then the diagonal element is an eigenvalue. If the eigenvalue happens to be ill-conditioned (sensitive to small changes in the matrix elements), it will have relatively large errors when determined by the routine *hqr*. We could have determined the isolated eigenvalue exactly and then deleted the corresponding row and column from the matrix.

4.3.2 Reduction to Hessenberg Form

First we reduce the matrix to a simpler form, and then we perform an iterative procedure on the simplified matrix. The simpler structure we use here is called Hessenberg form. An upper Hessenberg matrix has zeros everywhere below the diagonal except for the first sub-diagonal row. For example, in the 6×6 case, the nonzero elements are:

$$\begin{bmatrix} \times & \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times & \times \\ & \times & \times & \times & \times & \times \\ & & \times & \times & \times & \times \\ & & & \times & \times & \times \\ & & & & \times & \times \end{bmatrix}$$

Here we use a procedure analogous to Gaussian elimination with pivoting. Before the

rth stage, the original matrix $A \equiv A_1$ has become A_r, which is upper Hessenberg in its first $r-1$ rows and columns. The rth stage then consists of the following sequence of operations:

- Find the element of maximum magnitude in the rth column below the diagonal. If it is zero, skip the next two "bullets" and the stage is done. Otherwise, suppose the maximum element was in row r'.
- Interchange rows r' and $r+1$. This is the pivoting procedure. To make the permutation a similarity transformation, also interchange columns r' and $r+1$.
- For $i = r+2, r+3, \ldots, N$, compute the multiplier

$$n_{i,r+1} \equiv \frac{a_{ir}}{a_{r+1,r}}$$

Subtract $n_{i,r+1}$ times row $r+1$ from row i. To make the elimination a similarity transformation, also add

$n_{i,r+1}$ times column i to column $r+1$.

A total of $N-2$ such stages are required.

4.3.3 The QR Algorithm for Real Hessenberg Matrices

The basic idea behind the QR algorithm is that any real matrix can be decomposed in the form

$$A = Q \cdot R, \qquad (17)$$

where Q is orthogonal and R is upper triangular. Now consider the matrix formed by writing the in the opposite order:

$$A' = R \cdot Q \quad . \qquad (18)$$

Since Q is orthogonal, equation (17) gives $R = Q^T \cdot A$. Thus equation (18) becomes

$$A' = Q^T \cdot A \cdot Q. \qquad (19)$$

We see that A' is an orthogonal transformation of A.

The QR algorithm consists of a sequence of orthogonal transformations:

$$A_s = Q_s \cdot R_s, \qquad (20)$$

$$A_{s+1} = R_s \cdot Q_s \quad (= Q_s^T \cdot A_s \cdot Q_s). \qquad (21)$$

The QR transformation preserves the upper Hessenberg form of the original matrix $A \equiv A_1$, and the workload on such a matrix is $O(n^2)$ per iteration as opposed to $O(n^3)$ on a general matrix. As $s \to \infty$, A_s converges to a form where the eigenvalues are either isolated on the diagonal or are eigenvalues of a 2×2 sub-matrix on the diagonal.

In order to accelerate convergence, we deployed the technique of shifting: If k is any constant, then $A - kI$ has eigen-values $\lambda_i - k$. If we decompose

$$A_s - k_s I = Q_s \cdot R_s, \qquad (22)$$

so that

$$\begin{aligned} A_{s+1} &= R_s \cdot Q_s + k_s I \\ &= Q_s^T \cdot A_s \cdot Q_s \end{aligned}, \qquad (23)$$

then we verified that the convergence is determined by the ratio

$$\frac{\lambda_i - k_s}{\lambda_j - k_s}$$

here $\lambda_i < \lambda_j$. A good choice for the shift k_s would maximize the rate of convergence.

Any real matrix can be triangularized by pre-multiplying it by a sequence of Householder matrices P_1 (acting on the first column), P_2 (acting on the second column),..., P_{n-1}. Thus $Q = P_{n-1} \cdots P_2 \cdot P_1$, and the first row of Q is the first row of P_1 since P_i is an $(i-1)\times(i-1)$ identity matrix in the top left-hand corner.

The Householder matrix P_1 is determined by the first column of $(A_s - k_{s+1}I)\cdot(A_s - k_s I)$, which has the form $[p_1, q_1, r_1, 0, \ldots, 0]^T$, where

$$
\begin{aligned}
p_1 &= a_{21}[\frac{(a_{nn}-a_{11})(a_{n-1,n-1}-a_{11})-a_{n-1,n}a_{n,n-1}}{a_{21}}+a_{12}] \\
q_1 &= a_{21}[a_{22}-a_{11}-(a_{nn}-a_{11})-(a_{n-1,n-1}-a_{11})] \qquad . \qquad (24) \\
r_1 &= a_{21}a_{32}
\end{aligned}
$$

Since it has only first 3 elements nonzero, the matrix $P_1 \cdot A_s \cdot P_1^T$ is upper Hessenberg with 3 extra elements:

$$
P_1 \cdot A_s \cdot P_1^T = \begin{bmatrix}
\times & \times & \times & \times & \times & \times & \times \\
\times & \times & \times & \times & \times & \times & \times \\
X & \times & \times & \times & \times & \times & \times \\
X & X & \times & \times & \times & \times & \times \\
 & & & \times & \times & \times & \times \\
 & & & & \times & \times & \times \\
 & & & & & \times & \times
\end{bmatrix}.
$$

This produces a matrix with the 3 extra elements appearing one column over:

$$
\begin{bmatrix}
\times & \times & \times & \times & \times & \times & \times \\
\times & \times & \times & \times & \times & \times & \times \\
 & \times & \times & \times & \times & \times & \times \\
 & X & \times & \times & \times & \times & \times \\
 & X & X & \times & \times & \times & \times \\
 & & & & \times & \times & \times \\
 & & & & & \times & \times
\end{bmatrix}.
$$

Proceeding in this way up to P_{n-1}, we see that at each stage the Householder matrix P_r has a vector that is nonzero only in elements *r*, *r+1*, and *r+2*. These elements are determined by the elements *r*, *r+1*, and *r+2* in the *(r−1)*st column of the current matrix.

In summary, to carry out a double QR step we construct the Householder matrices, P_r, $r = 1, \ldots, n-1$. For P_1 we use p_1, q_1, and r_1 given by (24). For the remaining matrices, p_r,

q_r, and r_r are determined by the *(r, r−1), (r+1, r−1),* and *(r + 2, r − 1)* elements of the current matrix.

Algorithm description:

There are two possible ways of terminating the iteration for an eigenvalue. First, if $a_{n,n-1}$ becomes "negligible," then $a_{n,n}$ is an eigenvalue. We can then delete the nth row and column of the matrix and look for the next eigenvalue. Alternatively, $a_{n-1,n-2}$ may become negligible. In this case the eigenvalues of the 2×2 matrix in the lower right-hand corner may be taken to be eigenvalues. We delete the nth and *(n−1)*st rows and columns of the matrix and continue.

The test for convergence to an eigenvalue is combined with a test for negligible sub-diagonal elements that allows splitting of the matrix into sub-matrices. We find the largest *i* such that $a_{i,i-1}$ is negligible. If $i = n$, we have found a single eigenvalue. If $i = n-1$, we have found two eigenvalues. Otherwise we continue the iteration on the sub-matrix in rows *i* to *n*.

After determining *i*, the sub-matrix in rows *i* to *n* is examined to see if the product of any two consecutive sub-diagonal elements is small enough that we can work with an even smaller sub-matrix, starting say in row m. We start with $m = n-2$ and decrement it down to $i+1$, computing *p, q,* and *r* according to equations (24) with 1 replaced by *m* and 2 by *m+1*. If these were indeed the elements of the special "first" Householder matrix in a double QR step, then applying the Householder matrix would lead to nonzero elements in positions *(m+1,m−1), (m+2,m−1),* and *(m+2,m).* We require that the first two of these elements be small compared with the local diagonal elements $a_{m-1,m-1}$, $a_{m,m}$ and $a_{m+1,m+1}$. A satisfactory approximate criterion is

$$|a_{m,m-1}|(|q|+|r|) \ll |p|(|a_{m+1,m+1}|+|a_{m,m}|+|a_{m-1,m-1}|)\,. \qquad (25)$$

If ten iterations occur without determining an eigenvalue, the usual shifts are replaced for the next iteration by shifts defined by

$$\begin{aligned} k_s + k_{s+1} &= 1.5\times(|a_{n,n-1}|+|a_{n-1,n-2}|) \\ k_s k_{s+1} &= (|a_{n,n-1}|+|a_{n-1,n-2}|)^2 \end{aligned}\,. \qquad (26)$$

This strategy is repeated after 20 unsuccessful iterations. After 30 unsuccessful iterations, the routine reports failure.

Because our system carries out exact arithmetic operations, we are able to set the tolerance for the condition as "negligible", which means using $a_{n,n-1} < Tolerance$ and

$$\frac{|a_{m,m-1}|(|q|+|r|)}{|p|(|a_{m+1,m+1}|+|a_{m,m}|+|a_{m-1,m-1}|)} < Tolerance, \quad \text{instead of } a_{n,n-1} = 0 \quad \text{and}$$

$$(\mathit{float})(|a_{m,m-1}|(|q|+|r|)+|p|(|a_{m+1,m+1}|+|a_{m,m}|+|a_{m-1,m-1}|)) == |p|(|a_{m+1,m+1}|+|a_{m,m}|+|a_{m-1,m-1}|)$$

when deploying floating point arithmetic. The results would be more precise with a smaller tolerance.

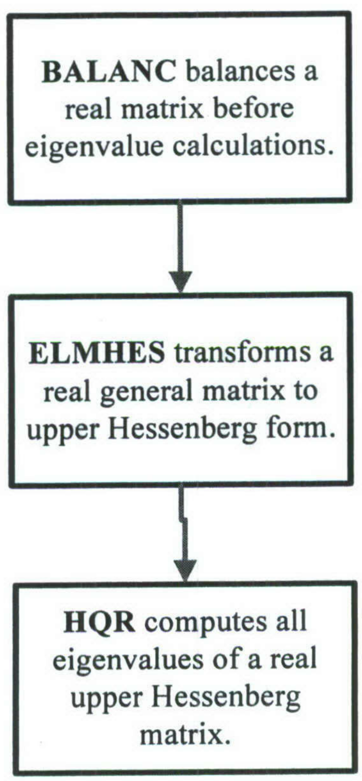
4.3.4 The Modules

Our program calculates the eigenvalues of an $N \times N$ real general Matrix A. The algorithm is a translated version of the EISPACK subprogram *RG.F*.

List of Routines:

RG.F calls subroutines BALANC, ELMHES and HQR.

Program Overview Flowchart:



BALANC

void balanc(vec_ZZ & vecTnum, vec_ZZ & vecTden,mat_ZZ & matPadics1,int & nm, int & n,int & low,int & igh,vec_ZZ & scaleNum, vec_ZZ & scaleDen);

Function Description: It balances a real matrix and isolates eigenvalues whenever possible.

INPUT:

nm must be set to the row dimension of two-dimensional array parameters as declared in the calling program dimension statement.

n is the order of the matrix.

vecTnum vecTden and matPadics1 contain the input matrix to be balanced.

OUTPUT:

vecTnum vecTden and matPadics1 contain the balanced matrix.

low and *igh* are two integers such that *A[i,j]* is equal to zero if (1) *i* is greater than j and (2) *j=1,...,low-1* or *i=igh+1,...,n.*

scale contains information determining the permutations and scaling factors used.

ELMHES

void elmhes(vec_ZZ & vecTnum, vec_ZZ & vecTden,mat_ZZ & matPadics1,int & nm, int & n,int & low,int & igh ,int & inte);

Function Description: Given a real general matrix, it reduces a sub-matrix situated in rows and columns low through igh to upper Hessenberg form by stabilized elementary similarity transformations.

INPUT:

nm must be set to the row dimension of two-dimensional array parameters as declared in the calling program dimension statement.

n is the order of the matrix.

low and *igh* are integers determined by Balanc. If Balanc has not been used, set *low=1, igh=n.*

vecTnum vecTden and matPadics1 contain the input matrix.

OUTPUT:

vecTnum vecTden and matPadics1 contain the Hessenberg matrix. The multipliers, which were used in the

reduction, are stored in the remaining triangle under the Hessenberg matrix.

inte contains information on the rows and columns interchanged in the reduction. Only elements *low* through *igh* are used.

HQR

void hqr(int & nm, int & n, int & low,int & igh,vec_ZZ & vecTnum, vec_ZZ & vecTden,mat_ZZ & matPadics1,vec_ZZ & wrNum,vec_ZZ & wrDen,vec_ZZ & wiNum,vec_ZZ & wiDen,int & ierr);

Function Description: It finds the eigenvalues of a real upper Hessenberg matrix by the QR method.

INPUT:

nm must be set to the row dimension of two-dimensional array parameters as declared in the calling program dimension statement.

n is the order of the matrix.

low and *igh* are integers determined by Balanc. If Balanc has not been used, set *low=1, igh=n.*

vecTnum vecTden and matPadics1 contain the upper Hessenberg matrix. Information about the transformations used in the reduction to Hessenberg form by Elmhes, if performed, is stored in the remaining triangle under the Hessenberg matrix.

OUTPUT:

vecTnum vecTden and matPadics1 have been destroyed. Therefore, it must be saved before calling *hqr,* if subsequent calculation and back transformation of eigenvectors is to be performed.

wrNum wrDen and *wiNum wrDen* contain the real and imaginary parts, respectively, of the eigenvalues. The eigenvalues are unordered except that complex conjugate pairs of values appear consecutively with the eigenvalue having the positive imaginary part first. If an error exit is made, the eigenvalues should be correct for indices *ierr+1,...,n.*

ierr is set to zero for normal return, or set to *j* if the limit of *30*n* iterations is exhausted while the *j*-th eigenvalue is being sought.

The matrix and related parameters are stored as p-adic sequences in mat_ZZ and fractional number. For the rational data structure, the numerator and denominator of the rational are stored separately in array

vecTnum and vecTden as arbitrary length integers. This maintains the precision of the numbers during the calculation. During computational process all rational numbers will keep their fractional data type. The floating-point data type was never used, to keep the calculation free of round-off/truncation errors.

Note the Error Code output: if normal return, *ierr* = *-1*; if Error Code > *0*, it indicates that more than 30 iterations of a subroutine were required to determine an eigenvalue. In this case, the subroutine terminated after 30 iterations.

The eigenvectors are outputted as a square $N \times N$ matrix whose entries correspond to the eigenvalues as follows: If the *i*-th eigenvalue is real, the *i-th* COLUMN of the eigenvector matrix contains the corresponding eigenvector; If the *i*-th eigenvalue is complex with positive imaginary part, COLUMNS *i* and *(i+1)* of the eigenvector matrix contain the real and imaginary parts of the corresponding eigenvector.

4.3.5 Result Analyses

We use the following example to show the advantages of our programs in terms of exactness.

The input matrix $\begin{bmatrix} -\frac{4}{111} & \frac{14}{111} & 0 \\ -\frac{5}{111} & \frac{13}{111} & 0 \\ -\frac{1}{111} & 0 & \frac{2}{111} \end{bmatrix}$ has eigenvalues $\lambda_1 = \frac{2}{111}$, $\lambda_2 = \frac{6}{111}$, and $\lambda_3 = \frac{3}{111}$.

Result:

THE BALANCED MATRIX is:

2/111 0/1 -1/111

0/1 13/111 -5/111

0/1 14/111 -4/111

THE UPPER HESSENBERG MATRIX is:

2/111 0/1 -1/111

0/1 13/111 -5/111

0/1 14/111 -4/111

Eigenvalue 1:

2/111+ j0/1

Eigenvalue 2:

44032022843120654290378435526868383433715773553170377747095358400281485847264778

38853083241173929305733492217313050558132589976712796782174639787119558918188959

22170272007433966278435458954982246587849564662126277527673707724488387314845240

73166119448678611740344306722445319604100446455711872024503408794989428024999557

72367374331500429977063062653905440955495027034700110517834687322583943548114289

20487384484543724036357178869514787915773160244610311077901974476456747955678844

30047767278803391025832434833570222839488557098550613078318596870081849669675244

73135639084829788496287548762380218651169899910814485161617327888699986994382161

08831024771732166579722709412952864676956358714369091585502277049791467244315516

33288391380146088376156008608299607038932907728199605690521018011735767815276798

59094332359330079481467459662823318370479626325472496352345432847465969088296104

28373211047627295025895114939200921720177457521662696667452584613428566709527512

18032601247038636424675563928171582647205470537422467658488808855048852276404241

15665511824695446031097211301380086764455157186467712955273234825991280793288313

38731004861252085123399344975183826445596327774227596479231001879467647971989059

85705324337204429410057744644133151184503184321613724648787673213506174227469332

53386680227576456134520568147816008384575029455486717470583889955442089675305851

35956178819028560493142657814827072103071061003638462512403630315276086657439555

78853367947057069029770166333602879891190733134152298150652563067540320404704145

78228990148057665095210817347368915696729714546326862223739233814284452302608771

59737283720351212204287788577980070266208143665421741015899903767280098068319985

16001292304632065514737462643825821745396790130776227656372388476629287961434674

14325788357161342537827867288662463808357178263932301596598063554548227661025957

24439199876100321325798196126737664124616401696776542557373577409745580597249478

97087349658537048770470310392011659863661909521784470714573456328028934513404199

30608111728960423077720099236589348239679428609695690342515610380878344966858032
04140223657424510114298383581170479103307987222089797995959058963603843466968175
732226750671036838008333496440823855132850977951726738381886443521/8145920243782
43286698659646628692192984982513568068667732824273057509259544569876376010302129
66467041038894305621459903039222541390787332954482100506309809838393635973213081
66079968918484828514581685649477801312200555078303900637798083185219291906305311
23684901809920954542967715335547269399378597158706324515889726366716183278863345
25617984099218717269159098033106919806346215646563968489653731895013754521469132
99324138603198176950531003791820971646699862123473123343263389197453818707126769
74538664850848559048855444059874821982236317455864428113775876713388646225147111
22443460956038709017149353546276946114981866591214549752747253351428145542453341
04112587479588894604427105664523299000439612604318505747345365619730693766391724
30457710640595101071561042006094598319606565938417611133067664188736335746269603
47217018237629144647769407331047628729657501120721717001979827504744651812511147
77306329148716635243562438340770545006545492735143929314507702854422264288277652
90208206058948196325108270367107261265258746413352755732272437359618325737476761
29821217405840016473707328622006677499431976340893306600785871480643767951494167
23211208919633500477361312883327899264268485979531471532109205898377631543296859
69124808475984597591306538288848265812090545658906591396610597676687826263821894
53452156149574076292816566841706836609683407052442220432221453441998099696529593
39804726317722842121189777843593316564989925883254903846988982682944839850835487
59283215274590886048818399409490826010145312968402164085993658489747629840189356
20821965365031638465684413461095636229167671162136793103060198187229402927869201
31841212474183197221663613049932306707498879717831089609725368687288333816007879
72966516514974389859745387783051481788526044206996095620191371090402633048892058
02607758563696118773104861830078618683441931659217660545115892316829635871775756
39638392324954036506409791751153546361957415137723745351909267489592226233425431

6317885653902213753460323284444791924314175287345438834941385245913831611946848442681807604340724694107445125247221100439652494582522383618778441773705174597789

5372027567674785255803081255641579869865903673282257062489180207954243414801354968779901116559585476875154880951462289547353843257344O+ j0/1

Eigenvalue 3:

2201599204872033060972936535497280736632365069825951381789384830214461288218127181963681686863932079703089863077985934192115376193352022246848619771402315365203

0287446358499124981143806703128618912254041900115927171863702899185729471054118464586069381502139974761138330432070587754523717307653392222573546440136938850449

5897170419718756077596879098414280973608468087411053479795163358835240573527890660881096596559234262470257012153840396037365030506255562131703496498864982608338

4115298181330443195714625653701212545337505755732483653546771141333495733694298587943596433095662932017976972783881967542399454674394242376425228021490203761529

3663509391837165776425857191077258301543897386393346583343929926955220287681745604113777245519262592217409291326881775659565835124895340512921219631085354849813

6826589755704161660289484674190759116351614638147893074290013224184073570764124856686560229776086081330269968736885700821270693524428951584459643081711528760291

3461322679787433934285696650314718301913567105720067417291647740285392099179633902345491103014623700840318507276415921015764477291265562685668446681363752108057

3148682154063705190134446039643617889103687403135425014546336023871990828608591875276729547138347821254112203851181465899264835637260491354823467692500554304646

3942174650196985575919364391215270664131363882066740571805548174746581274112618909345781859599701438077312579902093904720723381512641531973408923265905341587131

7578765688571140778775394059074500677646795224598281753874824839177724450258541049069788645753082057528025951572385936017032347556079483695977993950692277787221

4401122209175782090699984483245659336658403653794139813373441946501492038520273888196425819183696489503040723597983239625324290046788582844898473459537368614312

0924127386174207633528546532815100336182724062722789545533338516707209295197259

0034477733702342144552355715108092185353592787219347820265186370479456334708618

5740079031436068649023809854177810053613307419816831050214824497957299227394714

2226760677671844772171433363791694701226945679931039390014214094333218941328908

6880151668843875531904329916576436120901738750137104259259256778285913302520578

528230013904272779343155738519307856670855747368531246603169169407681459226330

9365773026644929997159333082380918674186600858881874661281967224669906630963606

2172021100573213344234451564874377580683592153202944772296279158328048152894780

7834576782009883317087697993899588701265607792094773020762184280562598382017809

7820642785457730009602346916491745661615309108513201541890069680417772060900961

4650199390396829452580582452080428530937652342236548680776348768266156759082654

7550664590059647223943582665988071038331329095613059380072076637141532627850186

5074558526803660762490156959469337227046996963187710291540416840805719765459972

0596296931742067460272867694043997884867780405888253844709545389825593384607309

8686940699139100416185687564813910490970924441170026549842245581501235725179998

5937640595879335925542625489577136223434657911586669747202498152479588937693997

7027690515838539151979416752562712642365725114825932936730488110886554101514307

5738973190952624792276560693060867084585512058177467807828935568971541073340494

2449475963223388812346106921275658698954945077911193933351692430309241691668032

5713794321413814519084460169010474407913865940692638911435311540925361063221372

5874293533005826247314180712725694536320283138559334507037361714379154659083201

3568163750313593889145007633008852762371798198067353064771781639802662564270340

8800439556381360712748196955195096905464372863021553935800839772717493310144607

9529433644105908814289762552262144670129298580335984611033574906522471886641763

8371795165586826574114941519809341194945675974308109153319539556119172353539821

6884113429860852111812379990842697187198377519703500557986446382322564920096406

9511919525841264944674454001303676088643248283760642526529787058267506096698489

605113820191825102410519529017591044331537403416310528833172920361044043382064468

81194527331480535166259021749671017712192449490219251970337339431485726709989654

96942207328995862139063569994252025276440535290179874631670833646505346718918227

02651289483605420634326577978497485950488163291474699646093863977026762394818930

83776888573483781601940843849175645606233866292110823670307664350086514163466508

99355037898275689201553037174122448467622136983868352425261920417856760137613786

8384021496968048835052292594169661113324985454312552459 1+ j0/1.

As far as exactness is concerned, as we can see from the above example, our system can obtain much more accurate results, since floating-point arithmetic operations can only provide the precision limited by the precision of the data type. Our system performs much more accurate results and can be trusted in the process of exact computation.

5. Conclusions and Discussion

Despite the advantages of finite segment *p*-adic arithmetic, there is a problem need to be solved—the estimation of an efficient length *M* of the *p*-adic expansion in arithmetic operations. Currently, it cannot give the sufficient number of digits for *p*-adic sequence for exact computation in some situation. As we concluded that the maximum length of the *p*-adic expansion's periodical part of $(a \pm b)$ is $LCM(n,m)$; in multiplication, the maximum length of periodic part of the product of $(a \times b)$ is

$LCM(m,n) \times (p^{GCD(m,n)} - 1)$.

In order to compute matrix exponential with JCF, we performed eigenvalue analysis. The routines we used are translations of EISPACK: The matrix is first balanced; Orthogonal similarity transformations are used to reduce the balanced matrix to a real upper Hessenberg matrix; The implicit double-shifted *QR* algorithm is used to compute the eigenvalues and eigenvectors of this Hessenberg matrix.

Thus, to determine the efficient length *M* of the *p*-adic expansion for a rational number in the matrix operation becomes an important problem. In particular, the algorithm in eigensystems may use iterative arithmetic operations. This made the situation even more complicated. From our experience of

implementing the eigensystem in p-adic field, the mechanism for setting the length M is our biggest concern, because running time is related to it, a large M will increase the runtime tremendously. When dealing with iterative arithmetic operations, the required length of the p-adic sequence will keep on increasing during the computation.

One possible solution is to create a gradual increasing "M". As we know, the corresponding fraction numbers will be completely different for the same p-adic sequence with different lengths. If we can predict the range of the approximate eigenvalue based on previous results, we would be able to adjust the "M" when an abnormal result occurs. To be precise, we can set a judgmental mechanism after the operation, if an abnormal result occurs, redo the operation with an larger "M", at the same time, reduce the p-adic expansion of the products to its minimum size.

References

[1] Cleve Moler, Charles Van Loan "Nineteen Dubious Ways to Compute the Exponential of a Matrix, Twenty-Five Years Later", SIAM Review, Vol. 45, No. 1, 2003.

[2] Wilkinson, J.H., and Reinsch, C.1971, Linear Algebra, vol. II of Handbook for Automatic Computation (New York: Springer-Verlag).

[3] Smith, B.T., et al. 1976, Matrix Eigensystem Routines — EISPACK Guide, 2nd ed., vol. 6 of Lecture Notes in Computer Science (New York: Springer-Verlag).

[4] Shoup, V NTL library at: http://shoup.net/ntl/.

[5] Krishnamurthy, E. V., Rao, T. M., and Subramanian, K., "Finite-segment p-adic Number Systems with Applications to Exact Computation", Proceedings of the Indian Academy of Science, 81A, (1975), 58-79.

[6] Krishnamurthy, E. V., Rao, T. M., and Subramanian, K., "P-adic Arithmetic Procedures for Exact Matrix Computations", Proceedings of the Indian Academy of Science, 82A, (1975), 165-175.

[7] Rao, T. M., "Finite Field Computational Techniques for Exact Solution of Numerical Problems", Ph.D. dissertation, Applied Mathematics Department, Indian Institute of Science, Bangalore, 1975.

[8] Dixon, J. "Exact Solution of Linear Equations Using P-adic Expansions", Numerische Mathematik 40, 137-141 (1982) Springer-Verlag.